Tópicos B

Raphael Fernandes

Apresentação do problema

- Gerar uma matriz aleatoriamente na memória, após isso,transferir a matriz para GPU. Na GPU, criar uma nova matriz, onde linha i da nova matriz seja igual a linha i+1 da primeira matriz.

Especificações da placa

```
Nome: Tesla K80
Clock: 0.82 GHz
Número máximo de cada thread por bloco: 1024
Número de multiprocessadores no device: 13
Memória compartilhada SM: 114.688KB
Memória global: nan GB
Tamanho de warp em threads: 32
```

Como a matriz é representada?

- Suponha um array como a seguir:

- O array será usado para representar todos os elementos da matriz.

Como a matriz é representada?

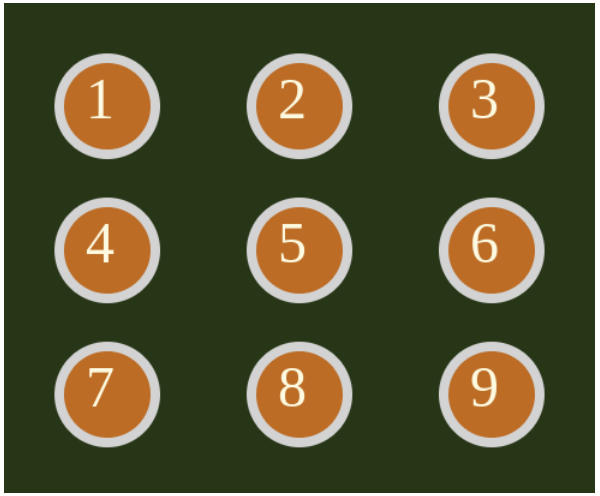
- Se abstraírmos que o array tem 1 dimensão, e imaginarmos que as colunas começam e terminam em uma determinada posição do array, por exemplo, de 3 em 3:

Como a matriz é representada?

- Se a matriz fosse representada com duas dimensões, M[L][C], por exemplo, o número de elementos seria dado por L*C.
- Assim sendo, para utilizar a matriz através de um array, os números de colunas determina em quais posições as linhas começam na matriz.

Posição de cada elemento

- 3 linhas e 3 colunas é igual a 9 elementos. O penúltimo elemento será L*C-1. A posição do número 6 será (L-1)*C, e assim sucessivamente.



Código sequencial

```
//Preenche a matriz de entrada aleatoriamente
for(int i = 0; i < len; i++){
    matrixInit[i] = (rand() % 99) + 1;
}
```

Código sequencial

```
void invertePosicaoDasLinhas(int* matrizEntrada, int* matrizSaida){

    int i, j;
    int elementoSaida, elementoEntrada;

    for(i=0; i<(L-1); i++){

        for (j = 0; j < C; j++){

            elementoSaida = i*C+j;
            elementoEntrada = (i+1)*C+j;
            matrizSaida[elementoSaida]=matrizEntrada[elementoEntrada];
        }

    }

    for(i=0; i < C; i++){

        elementoEntrada=i;
        elementoSaida=((L-1)*C)+i;

        matrizSaida[elementoSaida]=matrizEntrada[elementoEntrada];
    }
}
```

Código sequencial

```
Matriz de entrada:
29      44      73      80      24
71      56      40      70      2
42      41      6       26      96
5       43      55      80      56
99      9       61      34      27
```

```
Matriz de saída:
71      56      40      70      2
42      41      6       26      96
5       43      55      80      56
99      9       61      34      27
29      44      73      80      24

Tempo de execução:  0.00323 ms
```

Código CUDA

```
__global__ void invertePosicaoDasLinhas(int* matrizEntrada, int* matrizSaida) {

    int elementoSaida, elementoEntrada;

    //coluna de um elemento qualquer de matrixInit
    int j = blockDim.x * blockIdx.x + threadIdx.x;
    //linha de um elemento qualquer de matrixInit
    int i = blockDim.y * blockIdx.y + threadIdx.y;

    elementoSaida = i*C+j;
    elementoEntrada = (i+1)*C+j;
    matrizSaida[elementoSaida]=matrizEntrada[elementoEntrada];


    if(i==(L-1)) {

        elementoEntrada=j;
        elementoSaida=i*C+j;
        matrizSaida[elementoSaida]=matrizEntrada[elementoEntrada];
    }
}
```

Código CUDA

```
void invertePosicaoDasLinhas(int* matrizEntrada, int* matrizSaida){

    int i, j;
    int elementoSaida, elementoEntrada;

    for(i=0; i<(L-1); i++){

        for (j = 0; j < C; j++){

            elementoSaida = i*C+j;
            elementoEntrada = (i+1)*C+j;
            matrizSaida[elementoSaida]=matrizEntrada[elementoEntrada];
        }

    }

    for(i=0; i < C; i++){

        elementoEntrada=i;
        elementoSaida=((L-1)*C)+i;

        matrizSaida[elementoSaida]=matrizEntrada[elementoEntrada];
    }
}
```

```
__global__ void invertePosicaoDasLinhas(int* matrizEntrada, int* matrizSaida) {

    int elementoSaida, elementoEntrada;

    //coluna de um elemento qualquer de matin
    int j = blockDim.x * blockIdx.x + threadIdx.x;
    //linha de um elemento qualquer de matin
    int i = blockDim.y * blockIdx.y + threadIdx.y;

    elementoSaida = i*C+j;
    elementoEntrada = (i+1)*C+j;
    matrizSaida[elementoSaida]=matrizEntrada[elementoEntrada];


    if(i==(L-1)) {

        elementoEntrada=j;
        elementoSaida=i*C+j;
        matrizSaida[elementoSaida]=matrizEntrada[elementoEntrada];
    }
}
```

Resultados

- Sequencial
- Cuda